Continental J. Information Technology 5 (1): 12 - 21, 2011 ISSN: 2141 - 4033

http://www.wiloludjournal.com
` Printed in Nigeria

COST-EFFECTIVE MANAGEMENT OF COMPUTER SOFTWARE FOR IMPROVED PERFORMANCE USING REENGINEERING APPROACH

B. A. Ekanem
Computer Science Department Delta State Polytechnic, Ozoro. ba_ekanem@yahoo.com

ABSTRACT
The frequent maintenance of a software used over a long period of time say 5 to 10 years could render the software unmaintainable due to the unexpected and serious side-effects of successive maintenance and even the high costs of such exercise. When these occur, the software is either abandoned or replaced with another which may repeat the cycle and ends like its predecessor. In the alternative, some organizations may reengineer the software for improved quality and effectiveness. Therefore, this research was designed to perform a cost-effective analysis of the three options to guide management decisions on unmaintainable software. From the analysis conducted on data collected from 257 software in 64 organizations, it was observed that 70.73% of the software that replaced unmaintainable software were foreign while 29.27% were indigenous (developed locally in Nigeria), indicating lost of confidence in indigenous software due to unmaintainability. Also, three sampled software from the list exhibited at least 38.17% cost-saving advantage with software reengineering over other options of continuous maintenance or outright replacement with a new software, hence software reengineering is highly recommended.

KEYWORDS: Software Reengineering, Maintenance, Software Costs and COCOMO II.

INTRODUCTION

Computer software is one technology that plays major roles in our daily lives. Its application is found in every sphere of life especially banks, government parastatals, homes and private enterprises. The use of information-based systems driven by software technology is so fundamental to the operation and management of businesses that the introduction of a new technology usually results in changes to existing software if not outright replacement. Such changes are usually made through software engineering process called software maintenance which is usually ongoing to sustain the life of the software. Maintenance will correct errors found by users, adapt the product to a new environment, enhance its functionalities to satisfy users needs, and even institute changes in the procedure to increase effectiveness and efficiency (Shooman, 1983).

However, each time maintenance is performed on a software, a new version or release is produced. For instance, MS Office 2005, 2008 and 2007 are versions of initial MS office Suite while Windows 2000, XP, Vista and 7 are also versions of the initial MS Windows Operating System produced through properly managed maintenance processes. The largest expense on software systems is not the initial purchase of the software, but the complexity of implementing and maintaining the system. According to Banker et al (2002), since software maintenance is an ongoing process required to keep software useful, poorly managed maintenance can result in a steady stream of errors throughout the life of the software thereby making it ineffective and unmaintainable.

The frequency of maintenance of a software used over a long period of time say 5 to 10 years could render the software unmaintainable due to the unexpected and serious side-effects of previous maintenance and the high maintenance costs required in due course (Jewell 1991). To be on the safe side, most clients/user organizations normally ask for warranty periods of say 3 years where the vendor maintains the software free of charge or with a negligible fee (Ekanem, 2002). This fear is further confirmed by a Nigerian Warehouse software developer, Kenneth Okpeki who disclosed that one of the challenges he was facing is how to make people buy into his warehouse software (Ojiego, 2011).

Software Reengineering as a means of managing unmaintainable software can be described as a process of reorganizing and modifying existing software system to produce a higher quality and better maintainable system. Simply put, it is a rebuilding process for improved quality and effectiveness. In a perfect world, every unmaintainable program would be retired immediately, to be replaced with high-quality, reengineered software developed from modern software engineering practice and standard (Edelstein, 2003). But since we live in a world of unlimited resources, and software reengineering is likely to drain resources that could be used for other business purposes, there is need to ensure that the reengineering process for replacing an unmaintainable software system is cost-effective.

In view of the above, this research is designed to solve the problems associated with unmaintainable software systems with the view to providing a cost-effective means of restructuring such software based on modern software engineering practice to produce highly qualitative and maintainable software systems.

RESEARCH METHODOLOGY

The methodology for this research is as outlined below:

i. Review of existing relevant literature on software reengineering.
ii. Collection of data on unmaintainable software systems from client/user organizations as well as software developers and vendors using questionnaire, interview and observations. In this case, data were obtained on 257 software from 64 organizations (i.e. software organizations and users organizations).
iii. Cost-benefit Analysis was performed on the collected data using COCOMO II and Software Reengineering Models to determine whether software reengineering is most beneficial compared to other options of continuous maintenance or outright replacement with new software.
iv. Based on research findings and the cost-benefit analysis, some recommendations are made to guide software managers in dealing with unmaintainable software.

Review of Related Literature

In order to keep abreast with current trends in the research area, literature review was conducted with emphases on software maintenance and reengineering processes and cost-estimation models applicable.

a) Software Reengineering

Software reengineering is a process of restructuring an existing software to create a new version that exhibit higher quality and better maintainability. It can also be said to be a rebuilding activity just as the rebuilding of a house. Software reengineering is time consuming and capital intensive hence requires a pragmatic strategy by concerned organization to succeed (Pressman, 2005). The series of activities that make up this process include inventory analysis, document restructuring, reverse reengineering, program and data restructuring as well as forward engineering. The diagram below shows a cyclic model of the activities:

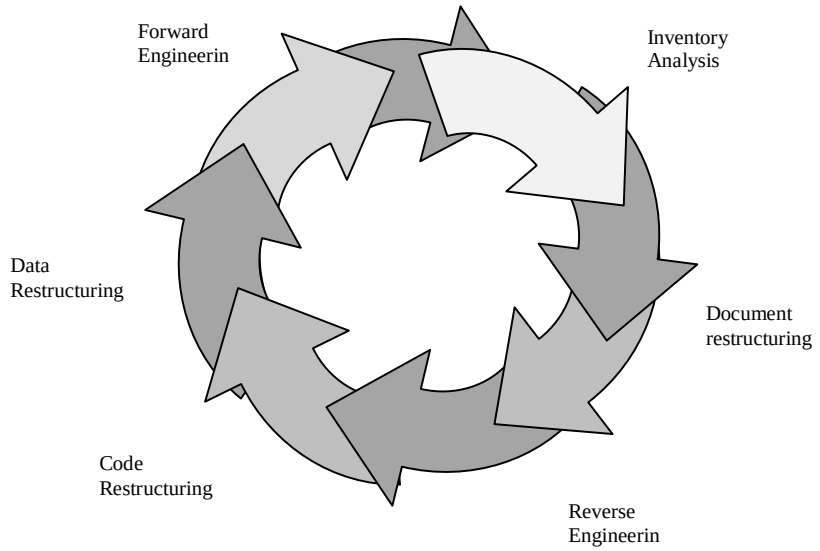

Fig. 1: Cyclic Model of Software Reengineering Process (Source: Pressman, 2005)

The first activity called Inventory Analysis has to do with assessing each application of concerned organization systematically with the intent of determining which are candidates for reengineering. Every software organization should have an inventory of all its applications showing detail description (e.g. size, age, business criticality) of every active application. By sorting this information according to business criticality, longevity, current maintainability and other important criteria, candidates for reengineering can be easily identified. It is important to note that the inventory should be revisited on a regular cycle since the status of application can change as a function of time, resulting in shift in reengineering priorities.

The second activity is Document Restructuring and has to do with creating a framework of documentation that is necessary for the long-term support of candidate software for reengineering. Successful document restructuring is followed by Reverse Engineering, the process of analyzing the candidate software in an effort to extract data, architectural, and procedural design information needed to understand the system and restructure it. Software reverse engineering is similar to its hardware counterpart where a company disassembles a competitive hardware product in an effort to understand its competitor's design and manufacturing "secrets" though in this case, the product (software) is not of competitor's but the company's own software to be reengineered by different set of software engineers due to obvious reasons like mobility of original software developers and the like.

With relevant information obtained through reverse engineering, Code Restructuring follows. As the name implies, Code Restructuring is the process of restructuring the code of program modules identified to be difficult to understand, test and maintain due to programmer's violation of structured programming constructs and software engineering standards. It is a process performed on a source code to yield a design that produces the same function as the original program but with higher quality and maintainability. To achieve this, the source code is analyzed using a restructuring tool to identify offensive modules which are then restructured automatically, and the restructured code reviewed and tested to ensure that no anomalies have been introduced. After this, the internal code documentation is updated accordingly.

With a successful code restructuring in place, Data Restructuring a process of restructuring the current data architecture used by the program to support software higher quality and maintainability follows. Finally, to complete the reengineering process, Forward Reengineering is performed by reconstructing the program using modern software engineering practice and knowledge acquired during reverse engineering. A careful and proper execution of this cyclic process should produce a software version with higher quality and better maintainability.

b) Software Reengineering Cost-Benefit Analysis Model

Software Reengineering requires funds to execute, hence the need for cost-benefit analysis of the process vis-à-vis other options to justify the investment. This is achievable with software reengineering cost-benefit analysis model presented by Sneed (1995) which is based on nine parameters given below:

P_1= current annual maintenance cost for an application
P_2= current annual operation cost for an application
P_3= current annual business value of an application
P_4= predicted annual maintenance cost after reengineering
P_5= predicted annual operations cost after reengineering
P_6= predicted annual business value after reengineering
P_7= estimated reengineering costs
P_8= estimated reengineering calendar time
P_9= reengineering risk factor (P_9 = 1.0 is nominal)
L = expected life of the system

From the above, the following costs values can be obtained:

i. (C_{maint}) the cost associated with the continuous maintenance of a candidate application (i.e. if reengineering is not performed) defined as

$$C_{maint} = [P_3 – (P_1 + P_2)] \text{ X } L \quad \text{---------- (eqn. 1)}$$

ii. (C_{reeng}) that is the cost associated with reengineering a candidate application defined as

$$C_{reeng} = [P_6 – (P_4 + P_5)] \text{ X } (L – P_8) - (P_7 \text{ X } P_9) \quad \text{---------- (eqn. 2)}$$

Using equations 1 and 2 above, the overall benefit of reengineering can be computed thus:

$$\text{Cost-benefit} = C_{maint} - C_{reeng} \quad \text{---------- (eqn. 3)}$$

For a software organization, the cost-benefit presented above can be performed for all high priority applications identified during inventory analysis and those that show high cost-benefit can be targeted for reengineering, while others can be postponed until resources are available. The model does not specify how to compute P_4 to P_9 hence relies on costs estimation models like Constructive Cost Model II (COCOMO II), Halstead, Expert Judgment, Function Point, and Lines-of-Code (Leung and Fan, 2002); (Jones, 2005); (Shen et al,1983).

c) Constructive Cost Model II (COCOMO II)

COCOMO is a family of models that was proposed and first published by Dr. Barry Boehm in 1981 and have been widely accepted in practice (Boehm, 1996); (Boehm and Faguhar, 1981). It is a model that allows one to estimate the cost, effort, and schedule when planning a software project. In the COCOMOs, the code-size S is given in thousand LOC (KLOC) and Effort is in person-month. Major versions of COCOMO include, Basic COCOMO, Intermediate COCOMO and COCOMO II which is the superset of the others is the latest.

COCOMO II which is short for COnstructive COst MOdel II is a model that allows one to estimate the cost, effort, and schedule when planning a software project. It consists of three submodels namely Applications Composition, Early Design, and Post-architecture models. COCOMO II came to being as a result of the dramatic change in software development techniques such as a move away from mainframe overnight batch processing to desktop-

based real-time systems. Also, increased emphasis on reusing existing software, problematic application of the original COCOMO and the need to incorporate professional software development practice were other reasons. The model uses seventeen cost factors with values ranging from 0.7 to 1.66, (Table 1).

The heart of COCOMO is based on the Effort Equation (eqn. 4), which applies a value to the tasks at hand based on the scope of the project (ranging from a small, familiar system to a complex system that is new to the organization).

Effort (in person-months) = a X EAF * $(KSLOC)^b$ ---------- (eqn. 4)

Where, coefficient a is 2.94 (approximately 3)

Scaling factor b, is 1 (i.e. 1.0997)

EAF is Effort Adjustment Factor obtained by multiplication of 17 parameters called cost drivers KSLOC is total physical size of all project files expressed in thousands Single Lines of Code

Table 1: COCOMO II Cost Factors and their weight

Cost Factor	Description	Rating				
		Very low	Low	Nominal	High	Very high
Product						
RELY	Required software reliability	0.75	0.88	1.00	1.15	1.40
DATA	Database size	-	0.94	1.00	1.08	1.16
CPLX	Product complexity	0.70	0.85	1.00	1.15	1.30
RUSE	Required Reusability	n/a	n/a	1.00	n/a	n/a
Computer						
TIME	Execution time constraint	-	-	1.00	1.11	1.30
STOR	Main storage constraint	-	-	1.00	1.06	1.21
PVOL	Platform volatility	-	0.87	1.00	1.15	1.30
Personnel						
ACAP	Analyst capability	1.46	1.19	1.00	0.86	0.71
APEX	Application experience	1.29	1.13	1.00	0.91	0.82
PCAP	Programmer capability	1.42	1.17	1.00	0.86	0.70
PLEX	Platform experience	1.21	1.10	1.00	0.90	-
LTEX	Language and Tools experience	1.14	1.07	1.00	0.95	-
PCON	Personnel Continuity	n/a	n/a	1.00	n/a	n/a
Project						
TOOL	Software tools	1.24	1.10	1.00	0.91	0.83
SCED	Development schedule	1.23	1.08	1.00	1.04	1.10
SITE	Multisite Development	n/a	n/a	1.00	n/a	n/a
DOCU	Documentation match to lifecycle need	n/a	n/a	1.00	n/a	n/a

* n/a means not available

Source: Adapted from Stan (2005)

The Costar 7.0 package from Softstar Systems, Amherst, NH (www.softstarsystems.com) is an automated tool for COCOMO implementation. Perhaps the most significant difference from the early COCOMO models is that the exponent *b* changes according to the cost factors. According to Stan (2005) the cost of a software project can best be estimated in terms of the four essential COCOMO II parameters: Complexity, process, Team and tools.

COCOMO II can be used for the following major decision situations

i) Making investment/financial decisions involving a software development effort.
ii) Setting project budgets and schedules as a basis for planning and control.
iii) Deciding on or negotiating tradeoffs among software cost, schedule, functionality, performance or quality factors.
iv) Making software cost and schedule risk management decisions.
v) Deciding which parts of a software system to develop, reuse, lease, or purchase
vi) Making legacy software inventory decisions: what parts to modify, phase out, outsource, etc.(Stellman and Greenee, 2005).
vii) Setting mixed investment strategies to improve organization's software capability, via reuse, tools, process maturity, outsourcing, etc.
viii) Deciding how to implement a process improvement strategy, such as that provided in the SEI CMM.

Data Collection

Data on software maintenance and reengineering costs for this research were obtained on 257 software systems from 64 organizations sampled from Lagos, Port Harcourt, Warri, Abuja, Kaduna, and Bauchi. The data are represented in Tables 2 and 3.

Table 2: Data on Maintenance/Reengineering of Indigenous and Foreign Software

s/n	Indicator	Indigenous			Foreign		
		No. of software	No. of organizat-ions	Ratio	No. of software	No. of organizat-ions	Ratio
1	New Software Products that Replaced Unmaintainable Counterparts	94	41	2:1	nil	nil	nil
2	Reengineered Software	10	6	2:1	nil	Nil	nil
3	Upgraded Software version	39	22	2:1	35	28	1:1
4	Software abandoned due to high maintenance costs	13	13	1:1	23	nil	1:1
5	Software never maintained nor reengineered	Nil	nil	nil	nil	nil	nil
6	Foreign Software products replacing indigenous counterparts	62	29	2:1	nil	n/al	n/a
7	Indigenous software replacing Foreign counterparts	n/a	n/a	n/a	nil	nil	nil

* n/a means not applicable

Table 3: Annual Software Operation and Maintenance Cost

s/n	Indicator	Cost involved in replacing with New product (₦)		
		A	B	C
1	Initial Product Acquisition Cost	2M	1.6M	.4M
2	Current Annual Maintenance Cost of Initial product (P_1)	.15M	.24M	.11M
3	Current Annual Operations Cost of Initial product (P_2)	.1M	.7M	.1M

Cost Analysis Results and Discussion

The project size for A, B, C are 2156, 2542 and 1785 respectively.

Table 4: Software Characteristics based on COCOMO II

Cost Factor	Description	Product A		Product B		Product C	
		Rating	COCOMO II Value	Rating	COCOMO II Value	Rating	COCOMO II Value
Product							
RELY	Required software Reliability	Very high	1.40	Nominal	1.00	High	1.15
DATA	Database size	High	1.08	High	1.08	High	1.08
CPLX	Product Complexity	High	1.15	Nominal	1.08	Low	0.85
RUSE	Required Reusability	Very high	1.30	Nominal	1.00	high	1.15
Computer							
TIME	Execution time constraint	nominal	1.00	nominal	1.00	nominal	1.00
STOR	Main Storage constraint	Nominal	1.00	Nominal	1.00	Nominal	1.00
PVOL	Virtual machine volatility	nominal	1.00	nominal	1.00	nominal	1.00
Personnel							
ACAP	Analyst capability	High	0.86	High	0.86	Very high	0.71
APEX	Application experience	Nominal	1.00	High	0.91	Very high	0.82
PCAP	Programmer capability	High	0.86	High	0.86	High	0.86
PLEX	Virtual machine experience	High	0.90	Low	1.10	High	0.90
LTEX	Language experience	High	0.95	High	0.95	High	0.95
PCON	Personnel Continuity	nominal	1.00	nominal	1.00	nominal	1.00
Project							
TOOL	Software tools	Nominal	1.00	Nominal	1.00	High	0.91
SCED	Development Schedule	Very high	1.10	High	1.04	High	1.04
SITE	Multisite Development	Nominal	1.00	Nominal	1.00	Nominal	1.00
DOCU	Documentation match to Lifecycle need	nominal	1.00	nominal	1.00	nominal	1.00
EAF	Effort Adjustment Factor	n/a	1.572	n/a	0.853	n/a	0.491

Table 5: Software Reengineering Costs Estimates

Software	Effort (in person-month) from eqn. 4	Cost ($)	Cost (N)
A	10.12	20,240	2,934,800 approx. 2.94M
B	8.69	17,380	2,520,100 approx. 2.52 M
C	3.16	6,320	916,400 approx. .92M

Effort is computed using eqn. (4). Project cost is computed by multiplying effort with average monthly developers salary which is estimated as $2000.

Table 6: Estimates for Computation of Costs Associated with Reengineering based on Expected Software Life

s/n	Indicator	Amount (₦)		
		Software A	Software B	Software C
1	Current annual business value (P_3)	2.6M	1.8M	2.2M
2	Predicted annual maintenance cost after reengineering(P_4)	.1M	.15M	.25M
3	Predicted annual operations cost after reengineering(P_5)	.2M	.1M	.1M
4	Predicted annual business value after reengineering(P_6)	4.3M	2.5M	2M
5	Estimated reengineering Costs (P_7)	2.94M	2.52M	.92M
6	Estimated Reengineering calendar time(P_8)	8 Months	8 Months	6 Months
7	Reengineering risk factor (i.e. 1.0 for nominal)(P_9)	1.0	1.0	1.0
8	Expected Life of the Software (L)	15 years	12 years	15 years

Table 7: Summary of Costs

Software	Costs of Continuous Maintenance for 15yrs, C_{maint} (₦)	Cost Associated with Reengineering and 15yrs Maintenance, $C_{reeng.}$ (₦)	Cost-benefit analysis (i.e. $C_{maint} - C_{reeng.}$)	% Difference
A	35.25M	26.7M	10.19M	38.17
B	10.32M	8M	3.84M	48
C	29.85M	13.93M	15.92M	114.29

RESULTS INTERPRETATION AND DISCUSSIONS

i. From Table 2, out of the 94 new software products that replaced unmaintainable counterparts in 41 organizations sampled, 62 were foreign products while 32 were indigenous (i.e. developed locally in Nigeria) which translates into 65.95% foreign and 34.05%. In terms of organizations involvement, 29 out of 41 replaced unmaintainable software with foreign counterparts which is 70.73% while 29.27 are indigenous. This implies that 70.73% of organizations have lost confidence in indigenous software due to unmaintainability.

ii. From Table 7, costs of continuous maintenance of software A for the next 15 years is a very high figure of ₦35.25M compared to 26.7M associated with reengineering within the period under review. This is because, after each maintenance within the period, the software complexity is likely to increases while software quality decreases thereby making subsequent maintenance more difficult and expensive. These characteristic features are also observed with Software B and C.

iii. Also from Table 7, the cost-benefit for Software A shows cost saving of ₦10.19M (i.e. 38.17% cost saving) by the software reengineering/subsequent maintenance instead of continuous maintenance of the unmaintainable software. Moreover, through the reengineering process, the software quality is likely to be highly enhanced thereby reducing the need for maintenance. Same situation is observed with Software B and C.

RECOMMENDATIONS

i. The quality of indigenous software products should be improved through the use of modern software engineering practice and development tools for increased users' confidence.
ii. Software Reengineering should be adopted by organizations as a cost-effective means of managing unmaintainable software rather than replacing such with new software systems which may recycle the process and end like their predecessors.
iii. COCOMO II should be used to compute software project costs estimates to ensure effective planning and resource allocation during the project.
iv. Software Reengineering model should be used to compute cost-benefit analysis on software continuous maintenance and reengineering.

CONCLUSION

To keep computer software useful over a long period of time, there is need for maintenance. However, the frequency of software maintenance over a period of say 5 to 10 years could render it unmaintainable due to unexpected and serious side effects of previous maintenance and the high maintenance cost required thereafter. When this occurs, it is better to reengineer the software for improved quality and maintainability as observed in this research work rather than its continuous maintenance since it is already unproductive.

REFERENCES

Banker, R. D., Datar, S. M., Kemerer, C. F., Zweig, D. (2002). Software Errors and Software Maintenance Management. *Information Technology and Management Journal*, Kluwer Academic Publishers, Netherlands, 3:1-2, 25-41.

Boehm, B. W. (1996). *The COCOMO 2.0 Software Cost Estimation Model,* American Prgrammer, pp. 2 – 17

Boehm, B. W. and Faquhar (1981). *Software engineering economics*, Englewood Cliffs, NJ: Prentice-Hall, pp. 25, 40-52

Edelstein, D. V. (2003). International Software Engineering Standards will affect trade in the European Community, *Business America,* USA.

Ekanem, B. A. (2002). *Coping with Software Maintenance Cost with respect to warranty terms and manpower deployment: a case study of Creation Nigeria Limited* pp. 10-21

Jewell, T. K. (1991). *Computer Applications for Engineers*.John Wiley & Sons Inc., New York, pp. 117-234

Jones, C. (2005). Software Cost Estimating Methods for large Projects, *the Journal of Defense Software Engineering*, pp. 5, 12-17

Leung, H., Fan, Z. (2002). Software Cost Estimation www.citeseerx.ist.psu.edu/viewdoc Retrieval date October 5, 2010

Ojiego, N. (2011). Nigerian Develops Software for Warehouse Management, *Vanguard Newspaper*, 25:61348 pp. 34

Pressman, R. S (2005). *Software Engineering, A practitioner's Approach*, Sixth Edition, McGraw-Hill Companies Inc., New York pp. 873-880

Shen, V. Y. and Conte, S. D., Dunsmore, H. E. (1983). Software Science Revisited: a critical analysis of the theory and its empirical support, *IEEE Transactions on Software Engineering*, pp. 155-165

Shooman, M. L. (1983). *Software Engineering*, McGraw-Hill Books Co., Singapore, pp. 469-490

Sneed, H. (1995). Planning the Reengineering of Legacy Systems, *IEEE Software*, pp. 24-25

Stan, B. (2005). Software Project Cost Estimates using COCOMO II Model, The Code Project website.www.codeproject.com/kb/architecture/COCOMO2.aspx Retrieval date April 7, 2011

Stellman, A. and Greene, J. (2005).*Applied Software Project Management*. O'Reilly Media.www.stellman-greene.com/aspm/. Retrieval Date October 10, 2010.

ACKNOWLEDGEMENT
I wish to acknowledge the Lord Almighty for seeing me through this research exercise. Also, I wish to express my deep sense of gratitude to my family members especially my wife, Elizabeth for their understanding during the research. My sincere appreciation goes to my boss, Prof. B. O. Ejechi who had always encouraged and guided me on how to do quality research. A big thanks also go to my professional colleagues namely Mr. Moses Ekpenyong, Mr. Nseabasi Essien and Dr. Ekabua Obeteng Obi for their professional advise. I will ever remain grateful to you all and even others not explicitly acknowledged. Thank you and God bless.

Received for Publication: 20/05 /2011
Accepted for Publication: 30/07 /2011